PL-TR-96-2092

CLOUD SIMULATION USING HEFeS-HIERARCHICAL ENVIRONMENTAL FEATURE STRUCTURE

Albert R. Boehm and J. H. Willand

Hughes STX Corporation
Hanscom AFB, Massachusetts 01731-3010

11 April, 1996

Scientific Report No. 5

PHILLIPS LABORATORY
Directorate of Geophysics
AIR FORCE MATERIEL COMMAND
HANSCOM AIR FORCE BASE, MA 01731-3010

This technical report has been reviewed and is approved for publication

[illegible]
DONALD D. GRANTHAM, Chief,
Atmospheric Structure Branch
Contract Manager

[illegible]
DONALD A. CHISHOLM, Acting Director
Atmospheric Sciences Division

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** 11 April 1996	**3. REPORT TYPE AND DATES COVERED** Scientific Report # 5

4. TITLE AND SUBTITLE CLOUD SIMULATION USING HEFeS-HIERARCHICAL ENVIRONMENTAL FEATURE STRUCTURE	**5. FUNDING NUMBERS** F19628-93-C-0051 PE62101F PR6670 TA GS WU AC
6. AUTHOR(S) Albert R. Boehm and J. H. Willand	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Hughes STX Corporation 29 Randolph Road Hanscom Air Force Base, MA 01731-3010	**8. PERFORMING ORGANIZATION REPORT NUMBER** Hughes STX Scientific Report #5
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Phillips Laboratory 29 Randolph Road Hanscom AFB, MA 01731-3010 Contract Technical Manager: Donald D. Grantham/GPAA	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** PL-TR-96-2092
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release: distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

The goal is to rapidly simulate cloud scenes including radiances using a large variety of cloud structure associated with a given area and season. HEFeS uses a hierarchy of climate objects for nine different scales of motion: climate regime, planetary wave, synoptic feature, meso feature, cluster, cell, sheet, voxel, and droplet.

Rather than store all this information for viewing from different angles, the reproducibility property of pseudo-random number generators is used to index location and properties of each object. This "Stochastic Indexing" allows for millions of objects to be specified and retrieved with nearly zero storage. A backward running random number generator allows desired properties (e.g. 95% coldest temperature) to be generated.

Instead of using ray tracing methods to render a scene, radiometric properties are precalculated for each object under various lighting conditions and stored as prototype objects called morficons. These morficons are stretched to adjust for viewing perspective, exact lighting, and individual shapes. The resulting scene is consistent with climatology and the physics of the atmosphere.

14. SUBJECT TERMS Cloud simulation, Weather simulation, Simulation, Morficon, Stochastic Indexing			**15. NUMBER OF PAGES** 36
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** Unlimited

Table of Contents

1. THE OPERATIONS RESEARCH REQUIREMENT ____ 1

1.1. Definition of a Cloud ____ 2

1.2. Clouds For Simulation ____ 2

2. PHYSICAL CLOUD MODELS ____ 3

2.1. Parcel Method ____ 3

2.2. The Slice Method ____ 3

2.3. Stability Indexes ____ 4

2.4. CISK ____ 5

2.5. Numerical Weather Prediction and MOS ____ 5

3. CLOUD STRUCTURE ____ 6

3.1. Vertical Motion Generation ____ 7

3.2. Cloud Domains and Streets ____ 8

4. HEFeS CLOUD STRUCTURE ____ 9

4.1. Scales of Propagation ____ 10

4.2. HEFeS Objects ____ 12

5. STOCHASTIC INDEXING ____ 14

5.1. Pseudo Random Number Generators ____ 15

5.2. Backward Random Number Generation ____ 16

6. THE MORFICON ____ 19

7. RECAPITULATION ____ 21

8. REFERENCES ____ 22

9. Appendix A: documentation of computer functins uniran, narinu, and zforp. ____ 24

1. THE OPERATIONS RESEARCH REQUIREMENT

The purpose of a weather simulation in operations research is to quantify how well a system will work when impacted by weather. In a military operational simulation, numerous calculations of propagation are required. The large number is due to the number of sensors and targets and the fact that they can rapidly change location. In addition, many simulations require multiple views as seen by a land observer, a pilot, a radar, an infrared sensor, etc.

The goal here then is to rapidly simulate cloud scenes including radiance's using the large variety of cloud structure associated with a given area and season. Clouds are specified to be consistent with:

1. Known laws of physics.
2. Climate of designated area.
3. Multiple sensors, e.g. Visible, IR, Radar.
4. Purpose of the simulation.

1.1. Definition of a Cloud

A Cloud here is defined as a natural collection of liquid or frozen water droplets that degrades propagation or obscures vision.

1.2. Clouds For Simulation

There are three methods of providing clouds for use in operational simulation: 1) Archived observations. 2) Dynamically produced clouds. 3) Statistical and Stochastic models.

Archived observations are satisfactory for certain kinds of simulations but, in general, do not have adequate resolution in time and space. Further, many physical characteristics in archived data bases are not measured but inferred, such as, vertical motions, density, liquid water content, cloud tops (from inverting satellite-observed radiances), etc.

Current dynamically produced cloud models are a long way from producing satisfactory cloudscapes as shown in Section 2 below.

That leaves statistical and stochastic generators to produce simulated cloud scenes. There are a variety of such generators. Three general methods - spectral, fractal, and hierarchical objects - have produced scenes with noticeable realism. However, spectral methods de-emphasize the relations between wave phases and it is these relations that determine some of the more important statistical qualities such as mean distance between clouds. Fractal methods are too slow for real time generation.

Thus, a hierarchical object methodology was chosen and is explained in Section 4 below.

2. PHYSICAL CLOUD MODELS

There are five major types of physics-based models that relate physical fields and vertical profiles of temperature and humidity to cloud height and amount.

2.1. Parcel Method

The parcel method (Berry et al., 1945, p402) postulates what will happen to a parcel of air as it rises; at what level will saturation occur, how much, if any, vertical acceleration will occur. An important effect is that the heat of vaporization from water vapor condensing provides additional buoyancy.

2.2. The Slice Method

The slice or convective cell method (Berry et al, 1945, p693) takes into account the stability and thermodynamics of each layer for the entire column of air. The slice method is considered more reliable than the parcel method, but has some important limitations: 1) No net inflow or outflow from any stratum; 2) Initially Barotropic (density and pressure perfectly correlated); 3) Except for surface heating, no energy is transmitted into the column.

2.3. Stability Indexes

A large number of indexes (AWS/TR-79/006, 1979) have been developed. These are usually derived from physical reasoning but empirically adjusted according to observed cloud observations. The Showalter (1953) index which is designed to predict convective precipitation is an example of this type of index. Stability indexes provide a quantitative measure to the amount of turbulence and cumulus cloud formation. However, each has important assumptions and limitations. Some are pseudo-empirical in that input values to them are taken at certain altitudes known to provide good results from experience.

The above methods provide an estimate of cloud development based on various calculations of buoyancy in a column. They have proven to be useful in many cases. For example, the bases of clouds can be found using these methods. However, bases calculated by the computed convective condensation level (Petterssen, 1956, p153) are less then 0.8 correlated with observed cloud bases.

Furthermore, when these methods are applied to the tropical atmosphere, they often produce poor results.

2.4. CISK

To overcome these limitations, another mechanism was postulated, CISK or Convective Instability of the Second Kind as distinct from the column buoyancy forces of convective instability of the first kind. CISK (Atkinson, 1971, p2-10) hypothesizes interaction between energy of the buoyancy forces and meso-scale flow patterns which support each other.

2.5. Numerical Weather Prediction and MOS

Direct solution of the atmospheric fluid dynamic equations is limited by several factors. Integration of such equations require very large and fast computers. In order to run these integration's on today's computers, various assumptions and simplifications are used. However, a more fundamental problem is that the basic physical equations for stress and for water phase conversions (particularly with respect to ice crystals) are not completely known (Boehm,1994d). The undeveloped stress tensor is particularly discombobulating since it regulates the shape of cloud turrets, puffs, and streamers.

Numerical weather prediction (NWP) forecasts integrate the fluid dynamic equations of motion. However, these equations do not predict clouds nor rain directly through physical relationships. Instead various statistical relations most of which, like the Showalter index, are developed from physical reasoning but depend on empirical fitting for scaling, and on experience to judge their applicability.

The most general of these is the MOS (Model Output Statistics) developed from empirical relations between NWP model variables and observed clouds and precipitation (and many other variables of interest).

Perhaps a good summary of NWP and clouds is given by Dutton (1986). "Most of modern meteorological theory ignores the fact that the atmosphere contains water in its three phases. The theories of motion cannot deal with moist air because of the complexity of the phase change processes, and most of the theories of the phase changes themselves ignore the complexity introduced by atmospheric motion."

3. CLOUD STRUCTURE

The mean lifetime of an individual cloud droplet is of the order of 1 minute. Thus, a cloud is best thought of in terms of a droplet generating mechanism rather than a physical object. There are several droplet generating mechanisms, but the most dominate is condensation by cooling due to air rising. Air sinking shrinks a cloud.

3.1. Vertical Motion Generation

Since rising air can produce clouds and rain, it is useful to purvey the reasons for rising air. Vertical air motion is caused by thermals (heating of the surface producing bubbles of air that rise) or by horizontal air moving over a sloped surface which is either the ground or a sloped atmospheric structure such as a warm front.

Another ubiquitous cause of vertical motion is gravity waves. These are waves in the atmosphere physically similar to ocean waves. The term gravity refers to the restoring force in the wave; not to some waveform in the gravity field!

Gravity waves can propagate in any direction vertically or horizontally. The easiest kind to visualize are the waves along a surface between two atmospheric layers of different density. These waves look just like ocean waves; indeed the same shallow wave equation set is often used for both.

On the front edge of the wave, there is upward motion. At the rear, there is downward motion. These vertical motions are often enough to trigger the generation of a cloud or to suppress clouds. Rows of clouds along the upward motion edge of the wave are quite common. Indeed, gravity waves appear to be the dominate principle regulating cloud structure on the 100m to 50km scale.

Occasionally, clouds will line up straight as though drawn with a ruler, but the more common occurrence is for clouds to line up but with ragged edges. This effect can be modeled by using a wave field as the underlying probability density structure and then choosing random cloud centers based on this density.

There are times when there are no apparent cloud rows. Convective theory suggest that under calm conditions with no gravity waves that clouds will form in a regular hexagonal pattern like a bee's nest. The size between cells depends on layer thickness and instability. This condition does occur and has been photographed but is not the most frequent pattern.

3.2. Cloud Domains and Streets

The weather satellite images seen on TV show clouds massed along fronts and around weather features such as low pressure systems. While they do a satisfactory job of showing where clouds are, they only hint at the structure of clouds.

High resolution satellite imagery, shuttle photos, and high altitude aircraft photos show a wealth of structure.

Two types of features, domains and streets occur in nearly every scene. Cloud streets are another term for cloud rows.

Not all cloud rows are caused by gravity waves; some result from surface features such as mountains or surface hot spots which produce strong buoyancy updrafts. Note, however, that these features can be kinamatically specified by waveforms whether or not they are initiated by gravity waves.

The second type of feature, domains (Boehm, 1994b), are quite frequent. A cloud domain is an area usually 10 to 100 km across within which clouds appear the same; they have the same statistical structure.

4. HEFeS CLOUD STRUCTURE

Clouds are specified in HEFeS (Hierarchical Environmental Feature Structure) in an hierarchical fashion. Usually, the selection occurs from the top (largest stage) but can be initiated at any desired stage at the bottom or in the middle.

First, a synoptic regime is specified. This regime specifies the location of fronts and low pressure systems. Also, the vertical temperature and pressure profiles belonging to the selected regime are specified.

Next, based on the instability and availability of moisture as a function of altitude and synoptic climatology, cloud producing layers are specified by:

1. number of layers
2. base of layer
3. thickness of layer
4. fractional cloud amount
5. layer instability

HEFeS uses a hierarchy of climate objects for nine different scales of motion: climate regime, planetary wave, synoptic feature, meso feature, cluster, cell, sheet, voxel, and droplet. Selecting from this cascade of objects can rapidly generate a realization of the atmosphere including intricate detail.

4.1. Scales of Propagation

Meteorologists have for many years divided atmospheric motion into various scales depending on what forces are important at that scale. Various scales, from a single droplet (e.g. on a lens) to large domains as seen by a distant millimeter system, are required for effective propagation simulations.

Propagation is along a beam which is not infinitesimally thin but can often be described by a cone specified by some solid angle. The width of this cone at the target or at some atmospheric obstruction can be measured as the distance across the beam. There are beams which are very narrow (a centimeter or less in width), very short (less than 50 meters), very wide (10 km or more), or very long (500 km or more). What a sensor sees when looking through the atmosphere depends very much on its beam width and path length.

A meteorological satellite Infrared sensor may have a beam width of 3 km or more at the height of the clouds. A high resolution surface based infrared sensor may have a beam width of less than 5 meters when looking at the same cloud. What they see will not have the same emissivity nor structure. A realistic simulation must have the ability to provide propagation at various scales.

4.2. HEFeS Objects

HEFeS specifies objects at various scales - a cascade of objects as shown in Table 1:

TABLE 1. HEFeS Object Classes

OBJECT CLASS	TYPICAL SIZE	TYPICAL TIME	EXAMPLE
Climate regime	global	1 month	wet spring
planetary wave	5000km	1 week	zonal jet
synoptic feature	1000km	3 days	cold front
meso feature	100km	12 hours	squall line
cluster	1km	1 hour	cloud cluster
cell	30m	10 min	cloud cell
sheet	5m	5 min	rain streamer
voxel	1m	10 sec	droplet dist.
particle	1mm	1 sec	snow flake

For a given simulation, there is one master HEFeS object. This is usually climate regime. Each sub-object must be consistent with its parent object. For example, rain virga is prohibited from being attached to a clear sky object. In the selection process these prohibitions are enforced by object conditional probability. A zero probability completely prohibits a sub-object while high probability facilitates the likelihood of a sub object being attached.

HEFeS is a statistical description of the atmosphere. When a selection, an instantiation, is made from HEFeS it is bounded by the HEFeS relationships which in turn are distributional fits to the atmosphere. The HEFeS selection represents a virtual world in which everything is known exactly. How closely does this virtual world resemble the real world?

The structure in HEFeS is fitted to climatological distributions. For many standard weather variables, the climatological distributions are known quite well. For others, distributions are poorly known.

Not all variables are in HEFeS yet. For example, static electricity is not included. Further, subsets of HEFeS contain only variables that are significant for given classes of simulations. If additional variables are needed, they can be added either through a parasitic relation to known variable or by obtaining their distribution.

On the other hand, like the real world, HEFeS is not bounded in time or space. For example, suppose a selection is made for a limited region, e.g. California. Then part way through a simulation, it is found that an additional region is needed, e.g. Nevada. Using the master scenario seed, the region can be extended so that the atmosphere is consistent with the initial region. The same effect is true with respect to time.

The HEFeS virtual world is a selection consistent with the statistics, meteorological effects, and physical constraints of the real world, but a selection does not necessarily resemble the atmosphere at any particular historical date. However, HEFeS objects can be used as basis functions for objective analysis of a real world atmosphere (Boehm, 1995). This capability allows HEFeS to generate weather on a historical date (for example, D Day in Normandy) that is consistent with whatever observations there are. Additionally, this capability allows HEFeS to generate weather that is consistent with current observation for live play simulations.

5. STOCHASTIC INDEXING

Rather than store all this information for viewing from different angles, the reproducibility property of a pseudo-random number generators can be used to index location and properties of each object. This "Stochastic Indexing" allows for millions of objects to be specified and retrieved with nearly zero storage. A backward running random number generator allows desired properties (e.g. 95% coldest temperature) to be generated.

5.1. Pseudo Random Number Generators

A typical computer random number generator starts with a seed number and rapidly returns one or more numbers - a sequence - that appears to be random. However, if the same starting seed is used, the exact same sequence will result.

Many types of scene generating algorithms use this feature to produce the same scene by using the same seed. But if only a part of the scene is needed, the whole scene must be redone in order to obtain that section of the sequence pertinent to the desired part. If an inverse Fourier transform method is used, the whole random sequence must be calculated to get the random phases (the amplitudes are usually fixed to produce a desired spatial spectrum).

Stochastic Indexing goes one step further. It uses multiple starting seeds to generate what is needed at a point. The advantage of stochastic indexing is that the very large number of objects required for intricate detail can be indexed rapidly and without a huge data base. However, for this representation to be the same on different computers, a portable random number generator is required.

Uniran (Marse and Roberts, 1983) is a portable prime modulus multiplicative linear congruential generator. It uses a multiplication and modulus to generate the next random number,

$$\mathbf{Z_{i+1}=CZ_i \text{ MOD } M} \qquad (1)$$

where

$\mathbf{C = 630360016,}$

$\mathbf{M = 2^{31} - 1,}$

Z is a 32 bit integer.

C is factored to allow for only 32 bit integer arithmetic by breaking the modulus function into factors, thus

$$\mathbf{a\ b \text{ MOD } M = (a \text{ MOD } M)(b \text{ MOD } M) \text{ MOD } M} \qquad (2)$$

which allows a and b to be 32 bit integers otherwise their product can be too large for a 32 bit integer. *Uniran* generates the same numbers on PC's, SG's, Main frames, etc.

5.2. Backward Random Number Generation

Suppose instead of a random scene, a specified scene is desired. What is the scenario seed that causes the desired scene? Since the "random" number that is associated with the scene (e.g. the 95% cloudiest) can be calculated, the question becomes, what is the scenario seed value that causes this random number? This value could be a desired threshold such as the 95% worst rain rate or a specific value measured in a real situation either historically or in live play. To get to this value, the generator would have to be run backwards. In order to do this, first the value is converted to the random number that generated it. This procedure involves finding the inverse transform for a distribution. That is, given a value, what is the cumulative probability of that value? See Boehm (1976,p8) or Law and Kelton (1991,p465).

Next comes the harder part. Given the probability quantity, what is the scenario seed that would have caused this value? What is required is a backward random number generator that produces the same numbers as *uniran* but in reverse sequence. The backward running generator that duplicates *uniran* numbers is itself a linear congruential generator which we call *narinu,*

$$\mathbf{Z_{i-1} = C_b Z_i \text{ MOD } M} \tag{3}$$

where

C_b = 746061359 for the backward generator,

$M = 2^{31} - 1$.

Derivation:

A random number some distance along the sequence can be calculated directly by

$$Z_{i+k} = C^k Z_i \text{ MOD } M \tag{4}$$

This equation (Law and Kelton, 1991, p425) is exact but of limited use. The *uniran* generator cycles in $2^{31} - 1$ numbers, so that $2^{31} - 2$ is the number before the current number. Thus, the constant for a backward generator is,

$$C_b = C^K \text{ MOD } M \tag{5}$$

where

$$K = 2^{31} - 2.$$

Actually C_b was found by running a PC 50 hours or so until $Z = 1$. Then C_b is the previous Z.

It is not necessary to call *narinu* 1000 times to get a number 1000 times earlier in the sequence. Instead, a new C, $C_{b1000} = C_b^{1000}$ is the index 1000 numbers before. Thus, with a set of **C's** for various jumps in the sequence, any prior value can be obtained with only a few backward number generator calls.

FORTRAN, C, and BASIC versions of the computer functions *uniran* and *narinu* can be found in Appendix A. There is a PASCAL version of *uniran* in Law and Kelton (1991, p451). Also listed in the Appendix is the function *zforp.* This function can be used for computing the random number seed that equates to a given probability value.

6. THE MORFICON

The number of calculations required to go from a mathematical physical description to a rendered scene is very large. To do this number of calculations in real time for virtual displays or even in constructive simulations becomes prohibitively expensive in terms of computer time.

Instead of using ray tracing methods to render a scene, radiometric properties are precalculated for each object under various lighting conditions and stored as prototype objects called Morficons. These Morficons are stretched (morphed is the computer term) to adjust for viewing perspective, exact lighting, and individual shapes. The resulting scene is consistent with the climatology of the place and time and the physics of the atmosphere.

The Morficon consists of previously calculated portions of object transmission scenes - icons - that are stretched into the correct aspect and size for a particular viewing angle. A pallet of Morficons is needed to provide calculations for various sensors and atmospheric structures.

The Morficon is not merely a computer trick to produce fast images although it does appear to be one of the fastest ways to generate an image. Propagation calculations generally require various parameters to be integrated along a line of sight. Typically, a grid of these parameters is used to interpolate the parameters to spacing along the line of sight. These values are then summed to approximate the integration.

With a Morficon in the virtual world, the exact value of the distance through an object (e.g. a cloud) is known. Also, the exact value of transmission or other parameter is know. These values are summed up by adding Morficons to a target plane. Exact line-of-sight integration's - to within computer precision - result.

An important example is monochromatic transmission along a pencil beam line of sight,

$$I_s = I_t e^{\int_s^t K\,dx} \qquad (6)$$

I_s is the illumination received at sensor s
I_t is illumination at target t
K is absorption coefficient
x is along line of sight.

For n homogeneous or linear gradient objects

$$I_s = I_t e^{\sum_{i=1}^{n} K_i \Delta x_i} \qquad or$$

$$\ln(I_s) = \ln(I_t) + \sum_{i=1}^{n} K_i \Delta x_i \qquad \mathbf{(7)}$$

Δx is distance along line of sight through object i
K_i is $(K_{enter}+K_{exit})/2$ for linear gradient object i.

Usually, there is a tradeoff between speed and accuracy. With the Morficon both ultra-speed and accuracy result. Part of this advantageous situation is due to the prior calculation in making the Morficon palette.

7. RECAPITULATION

There are many kinds of simulations. Here, the focus is on the effect of clouds on systems. The simple view is to treat clouds statistics - height, thickness, water content, etc. - as constants. But clouds vary tremendously. Indeed it is possible, that no two clouds are alike.

Dynamic and physical methods of generating clouds have serious deficiencies . That is not to say physical methods should be discarded. Rather, those deficiencies must be acknowledged, worked on, and solved.

The HEFeS, Stochastic Indexing, Morficon approach is a powerful combination. These methods are quite general and adaptable to other environmental simulations and to more general simulation.

8. REFERENCES

Berry, F. A., Jr., Bollay, E., and Beers, N. R., 1945: Handbook of Meteorology, McGraw-Hill, NY, NY, 1068pp.

Boehm, A. R., 1976: Transnormalized Regression Probability, AWS-TR-75-259. Pub. by Air Weather Service (MAC), USAF

Boehm, A. R., 1994a: Stochastic Indexing, PL-TR-94-2012, Contract F19628-93-C-0051, Hughes STX. ADA283143

Boehm, A. R., 1994b: Visual Translucent Algorithm (VISTA), *Simulation, 62*, No. 2, 91-97.

Boehm, A. R., 1994c: Introduction to HEFeS (Hierarchical Environmental Feature Simulator), PL-TR-94-2013, Contract F19628-93-C-0051, Prepared by Hughes STX for the Phillips Lab., Directorate of Geophysics, Hanscom AFB, MA. ADA283244/2/DDM.

Boehm, A. R., 1994d: Methods of supplying Atmospheric Effects in Simulation, 11DIS Workshop on Standards for the Interoperability of Distributed Simulations, Vol 1 Position Papers, Institute for Simulation and Training, University of Central Florida, Orlando, FL, p161-164

Boehm, A. R., 1995: Objective Analysis Using a Hierarchy of Climate Features, Sixth International Statistical Climatology

Meeting, University College Galway, Ireland, 19-23 June, 1995.

Dutton, J. A., 1986: The Ceaseless Wind, Dover Publications, Inc., NY. 617pp.

Atkinson, F. D., 1971: Forecasters' Guide to Tropical Meteorology, AWS TR 240, Air Weather Service.

Showalter, A. K.,1953: A Stability Index For Thunderstorm Forecasting, Bull.Am.Meteorol.Soc., 34, No.6, 250-252 .

AWS/TR-79/006, 1979: The Use Of The Skew T, Log P Diagram In Analysis And Forcasting. Air Weather Service (MAC), Scott AFB, Illinois 62225.

Law, Averill M. and W. D. Kelton, 1991: Simulation Modeling & Analysis, Second Edition, McGraw-Hill, N.Y., N.Y., pp269-280.

Petterssen, S., 1956: Weather Analysis and Forcasting, Second Edition, Vol. 2. McGraw-Hill, N.Y., N.Y.

Appendix A

Documentation of computer functions *uniran, narinu,* and *zforp.*

Function *uniran* (Marse and Roberts, 1983) is the random number generator selected for use with simulation processes within the FASTPROP project. It is a prime modulus multiplicative linear congruential generator of the form a(i)=(630360016*z(i-1)) (mod(2**31-1)). (The constant 630360016 is the product of the variable's mult1 and mult2 in the program listing of function uniran below). The uniran function listed below is a slightly modified version of the original the only difference being the removal of the built-in default seeds and their management sections.

Function *narinu,* originated by A. Boehm (1995), is a backward random number generator to be used in conjunction with function *uniran.* Thus, it computes the previous seed of the random number generator *uniran.* It uses a prime modulus multiplicative linear congruential generator of the form Seed(i)=[Seed(i-1)*a] MOD p. Specifically, a=630360016 for the forward generator but a=746061395 to run backwards. (The product of f1 and f2 in function *narinu* below equals the constant a for the backward generator).

Function *zforp,* designed by A. Boehm (1995), gets the random number seed that equates to a given probability (p) where p of course is derived by *uniran.*

FORTRAN versions of the function's *uniran, narinu, and zforp* are listed immediately below.

```
real *4 function uniran(zi)
c   ================================================================
c   Prime modulus multiplicative linear congruential generator
c   z(i) = (630360016 * z(i-1)) (mod(2**31 - 1)), based on MARSE
c   and Roberts' portable random-number generator UNIRAN.
c   adapted in FORTRAN for use on the USAF/PL UNIX system by
c   J. Willand of Hughes STX on Apr 3, 1995.
c   ================================================================
    integer *4 b2e15, b2e16, hi15, hi31, low15, lowprd,
   &        modlus, mult1, mult2, ovflow, zi
    data mult1/24112/, mult2/26143/
    data b2e15,b2e16,modlus/32768,65536,2147483647/
c   ----------------------------------------------------------------
    hi15   = zi / b2e16
    lowprd = (zi - hi15 * b2e16) * mult1
    low15  = lowprd / b2e16
    hi31   = hi15 * mult1 + low15
    ovflow = hi31 / b2e15
    zi     = (((lowprd - low15 * b2e16) - modlus) +
   &        (hi31 - ovflow * b2e15) * b2e16) + ovflow
    if(zi.lt.0)zi = zi + modlus
    hi15   = zi / b2e16
    lowprd = (zi - hi15 * b2e16) * mult2
    low15  = lowprd / b2e16
    hi31   = hi15 * mult2 + low15
    ovflow = hi31 / b2e15
    zi     = (((lowprd - low15 * b2e16) - modlus) +
   &        (hi31 - ovflow * b2e15) * b2e16) + ovflow
    if(zi.lt.0)zi=zi + modlus
    uniran = (2 * (zi / 256) +1 ) / 16777216.0
    return
    end27
```

```
      integer *4 function narinu(zi)
c     =================================================================
c     Gets the previous seed of the random number generator uniran.
c     Uses a prime modulus multiplicative linear congruential gererator
c     of the form Seed(i)=[Seed(i-1)*a] MOD p. Specifically, a=630360016
c     for the foward generator but a=746061395 to run backwards. In both cases
c     p=(2**31 -1). Factors f1*f2 = a prevent overflow.
c       Based on Marse and Roberts, 1983: "Implimenting a Portable
c       FORTRAN Uniform (0,1) Generator", SIMULATION, 41, p135-139.
c     Usage: Obtain the prior seed in the sequence using the current Seed zi.
c     Originated in BASIC by A. Boehm and converted to FORTRAN by
c     J. Willand both of Hughes STX. 3/29/95.
c     =================================================================
      double precision z, z2, zt, moduls, f1, f2
      integer *4 intger, zi
      data moduls/2147483647.0D0/, f1/6235.0D0/, f2/119657.0D0/
c     -------------------------------------------------------------
      z      = zi
      z2     = z * f2
      intger = z2 / moduls
      z2     = z2 - intger * moduls
      zt     = z2 * f1
      intger = zt / moduls
      zt     = zt - intger * moduls
      intger = zt
      narinu = intger
      return
      end

      integer *4 function izforp(p)
c     Gets random number seed that equates to probability p.
c     Originated by A. Boehm in BASIC and converted to FORTRAN
c     by J. Willand HSTX 3/26/95.
      real *4 p
      real *8 con
      integer *4 izz
      data con/16777216.0D0/
c     -------------------------------------------------------------
      pb     = p
      izz    = (pb*con +1.0) * 128.0
      izforp = izz
      return
      end
```

The FORTRAN listing below is a main program written to test functions *uniran, narinu, and zforp*. Actual results from a typical test case using this program are documented at the end of the listing.

```
c      Program testran.f, Vers. 1.0, FORTRAN
c      4/3/95, J. Willand, test uniran, narinu, & izforp routines.
c  ==============================================================
       integer *4 zi, zp, oldzi, newzi, index
c  --------------------------------------------------------------
1111  print *, 'Input an integer seed like 3 or 804 for example'
       read(*,100) zi
100   format(i20)
       if(zi.lt.0) go to 1000
       oldzi = zi
       u     = uniran(zi)
       newzi = zi
       index = narinu(zi)
       print 101, oldzi, newzi, index, u
101   format(' oldzi=',i15/,' newzi=',i15/,' index=',i15/,' Ran=',f6.4/)
       zp    = izforp(u)
       print 102,zp
102   format(' Seed ',i15,' equates to Ran above.')
       go to 1111
1000  stop
       end
```

In a sample test case, an integer value of 80 was entered which prompted the program to produce the following results:

```
oldzi  =            80
newzi  = 1036677399
index  =            80
Ran = 0.4827
Number  1036677632 equates to Ran above.
```

CCC versions of functions *uniran, narinu, and zforp* are listed respectively as::

```
float uniran(long *zi)
{
static long modlus= 2147483647;
static long b2e15 = 32768;
static long b2e16 = 65536;
static long mult1 = 24112;
static long mult2 = 26143;
long low15, lowprd, ovflow, hi31, hi15;
float ans;

/* C version of  the  prime modulus multiplicative linear congruential
 generator  z(i) = (630360016 * z(i-1)) (mod(2**31 - 1)), based on
 MARSE and Roberts' portable random-number generator UNIRAN.
 Adapted in FORTRAN for use on the USAF/PL UNIX system by
 J. Willand of Hughes STX on Apr 3, 1995.  */

  hi15 = *zi / b2e16;
  lowprd = (*zi - hi15 * b2e16) * mult1;
  low15 = lowprd / b2e16;
  hi31 = hi15 * mult1 + low15;
  ovflow = hi31 / b2e15;
  *zi = lowprd - low15 * b2e16 - modlus + (hi31 - ovflow * b2e15) * b2e16 +
          ovflow;
  if (*zi < 0) *zi += modlus;
  hi15 = *zi / b2e16;
  lowprd = (*zi - hi15 * b2e16) * mult2;
  low15 = lowprd / b2e16;
  hi31 = hi15 * mult2 + low15;
  ovflow = hi31 / b2e15;
  *zi = lowprd - low15 * b2e16 - modlus + (hi31 - ovflow * b2e15) * b2e16 +
          ovflow;
  if (*zi < 0)   *zi += modlus;
  ans = ((*zi / 256 << 1) + 1) / (float)16777216.;
  return ans;
}
```

```
long narinu(long *zi)
{
   static double moduls = 2147483647.;
   static double f1 = 6235.;
   static double f2 = 119657.;
   double z, z2, zt;
   long intger;
/*   Gets the previous seed of the random number generator uniran.
     Uses a PMMLCG PRIME modulus multiplicative linear congruential
     generator of the form Seed(i)=[Seed(i-1)*a] MOD p. Specifically,
     a=630360016 for the foward generator but a=746061395 to run
     backwards. In both cases p=(2**31 -1). Factors f1*f2 = a prevent
     overflow.
     Based on Marse and Roberts, 1983: "Implimenting a Portable
     FORTRAN Uniform (0,1) Generator", SIMULATION, 41, p135-139.
     Usage: Obtain the prior seed in the sequence using the current Seed zi.
    Originated in BASIC by A. Boehm and converted to FORTRAN by
    J. Willand both of Hughes STX. 3/29/95. */

   z = (double) (*zi);
   z2 = z * f2;
   intger = (long) (z2 / moduls);
   z2 -= intger * moduls;
   zt = z2 * f1;
   intger = (long) (zt / moduls);
   zt -= intger * moduls;
   intger = (long) zt;
   return intger;
}

long zforp(float *p)
{
   static double con = 16777216.;
   float pb;
   long izz;

/*  ----------------------------------------------------------
   Gets random number seed that equates to probability p.
   Originated by A. Boehm in BASIC and converted to FORTRAN
   by J. Willand HSTX 3/26/95.
   ---------------------------------------------------------- */
   pb = *p;
   izz = (long) ((pb * con + (float)1.) * (float)128.);
```

```
    return izz;
}
```

Quick Basic versions of functions *uniran, narinu,* and *zforp* are listed below.

```
FUNCTION UNIRAN (ZI&)
'Prime modulus multiplicative linear congruential generator (PMMLCG)
'of the form Seed(i) =[Seed(i-1)*A] MOD P Specifically,
'ZI(I) = (630360016*ZI(I - 1))  (MOD(2**31 - 1)),
'based on Marse and Roberts' portable random-number generator UNIRAN.
'Marse,K. and Roberts,S. 1983: Implimenting a Portable FORTRAN Uniform (0,1)
'    Generator", SIMULATION, 41, p135-139
'Usage: obtain the next U(0,1) random number using seed ZI&.
'ZI& must be a 32 bit integer; UNIRAN is returned as a real (32 bit) number.
'WARNING ZI& is updated to next seed after call to function.
'To QBasic 15 Feb 1995 by A.Boehm
CONST MULT1& = 24112, MULT2& = 26143
CONST B2E15& = 32768, B2E16& = 65536, MODLUS& = 2147483647
HI15& = FIX(ZI& / B2E16&)
LOWPRD& = (ZI& - HI15& * B2E16&) * MULT1&
LOW15& = FIX(LOWPRD& / B2E16&)
HI31& = HI15& * MULT1& + LOW15&
OVFLOW& = FIX(HI31& / B2E15&)
ZI& = (((LOWPRD& - LOW15& * B2E16&) - MODLUS&) + (HI31& - OVFLOW& *
B2E15&) * B2E16&) + OVFLOW&
IF ZI& < 0 THEN ZI& = ZI& + MODLUS&
HI15& = FIX(ZI& / B2E16&)
LOWPRD& = (ZI& - HI15& * B2E16&) * MULT2&
LOW15& = FIX(LOWPRD& / B2E16&)
HI31& = HI15& * MULT2& + LOW15&
OVFLOW& = FIX(HI31& / B2E15&)
ZI& = (((LOWPRD& - LOW15& * B2E16&) - MODLUS&) + (HI31& - OVFLOW& *
B2E15&) * B2E16&) + OVFLOW&
IF ZI& < 0 THEN ZI& = ZI& + MODLUS&
UNIRAN = (2 * (ZI& / 256) + 1) / 16777216#
END FUNCTION
```

```
FUNCTION NARINU& (ZIN&)
'Gets the previous seed of the random number generator UNIRAN.
'Uses a PMMLCG but with a "Backwards" multiplier,
'(PMMLCG is Prime modulus multiplicative linear congruential generator.)
'of the form Seed(i) =[Seed(i-1)*A] MOD P Specifically,
' A=630360016 for the forward generator but A=746061395 to run backwards
'In both cases P = (2**31 - 1).  Factors F1#*F2#=A prevent overflow.
'based on Marse and Roberts' portable random-number generator UNIRAN.
'Marse,K. and Roberts,S. 1983: Implimenting a Portable FORTRAN Uniform (0,1)
'    Generator", SIMULATION, 41, p135-139
'Usage: obtain the prior seed in the sequece using the current seed ZI&.
'ZI& must be a 32 bit integer; NARINU& is returned as a 32 bit integer.
'a # after a variable indicates a double precision (64 bit)  real number.
'8 Mar 1995 by A.Boehm, 29 Mar 1995 superfluous an#= statement removed.
'11 April 1995 superfluous statement removed
CONST MODLUS# = 2147483647, Cback# = 746061395, F1# = 6235, F2# = 119657
Z# = ZIN&' NOTE ZIN& is not changed in NARINU
Z2# = Z# * F2#
Z2# = Z2# - INT(Z2# / MODLUS#) * MODLUS#
Zt# = F1# * Z2#
NARINU& = Zt# - INT(Zt# / MODLUS#) * MODLUS#
END FUNCTION


FUNCTION ZForP& (P)
'Gets Random number seed Z, that equates to probability P
'Seed is for UNIRAN random number generator
'9 Mar 1995 by A.Boehm
ZForP& = (P * 16777216# + 1) * 128
END FUNCTION
```